

컴파일러 과제6
보고서
(1) - 제작

20181755 이건희

2024년 11월

1. 개발 환경

기존과 동일한 wsl2 상의 Ubuntu 22 LTS, gcc 11, gdb 12이다.

```
kh@ThinkPad-T16g2: ~  
kh@ThinkPad-T16g2:~$ flex --version  
flex 2.6.4  
kh@ThinkPad-T16g2:~$ bison --version  
bison (GNU Bison) 3.8.2  
Written by Robert Corbett and Richard Stallman.  
  
Copyright (C) 2021 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
kh@ThinkPad-T16g2:~$
```

lex로 flex, yacc로 bison을 사용하였다.

2. 파일 및 수정사항

과제 5의 코드를 대부분 그대로 사용하되, 다음과 같은 수정사항이 있었다

2-1. 문법 수정

```
specifier = (S_KIND) S_NULL  
level = (int) 0  
address = (int) 0  
value = (int) 0  
init = (A_NODE *) NULL  
type = (A_TYPE *) NULL  
line = (int) 1
```

해당 줄은 매개변수로 받은 id->type->kind를 검사하는데 정의되지 않은 자료형을 사용했을 경우 id->type이 NULL이다. 이렇게 된다면 id->type->kind는 당연히 접근할 수 없고, segfault가 발생한다. 이를 방지하려면 id->type가 NULL인지 검사하는 코드가 필요하다.

```
638  
639  
640  
case 31:  
    printf(format: "undefined type for identifier %s", s);  
    break;
```

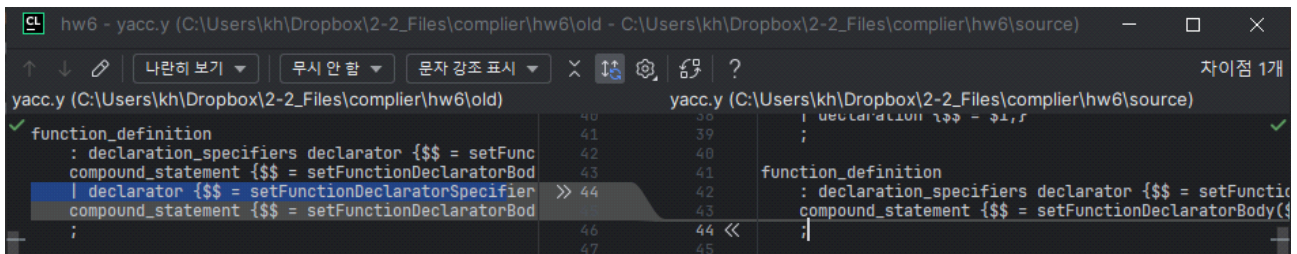
syntax_error()에서 정의되지 않은 자료형의 사용은 case 31이다. 어떤 정의되지 않은 자료형은 이전과 같이 잘못 파싱되었을 경우 id->name에 저장된다.

```
307  
308 // check function declarator and return type  
309 A_ID *setFunctionDeclaratorSpecifier(A_ID *id, A_SPECIFIER *p) {  
310     A_ID *a;  
311     // check storage class  
312     if (p->stor) {  
313         syntax_error(25, S_NULL);  
314     }  
315     setDefaultSpecifier(p);  
316  
317     if (id->type == NULL){  
318         syntax_error(31, id->name);  
319     }  
320 }
```

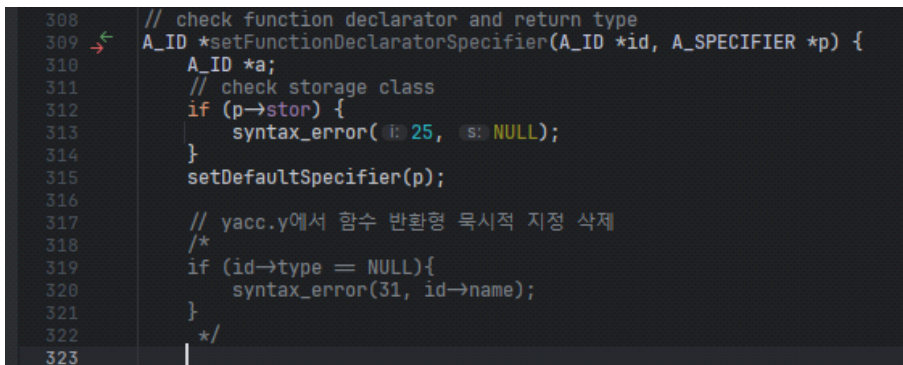
따라서 setFunctionDeclaratorSpecifier()의 사진과 같은 부분에

1. id->type이 NULL인지 검사하고
 2. NULL이라면 syntax_error(31, 정의되지 않은 자료형)
- 을 하는 코드를 추가했다.

지난번 과제의 보고서에 syntax_analysis.c의 setFunctionDeclaratorSpecifier()에서 오류가 나서 위와 같은 수정을 하였다고 적어서 냈다. 교수님과 줌으로 과제 설명을 하고나서 해결한 방법을 다시금 돌아보니 과제를 해결한 방법은 근시안적이라는 생각이 들었고, 교수님이 말씀하신 것처럼 문법 자체를 수정하는 것이 나아보여서 그렇게 수정하였다.



왼쪽이 이전에 사용했던 yacc.y, 오른쪽이 새로운 yacc.y이다. 기존의 yacc.y은 ANSI C의 규칙에 따라 자료형이 없으면 이를 묵시적으로 int형으로 지정하나, 새로운 yacc.y은 c99부터처럼 함수의 반환형을 묵시적으로 지정하지 못하게끔 하였다.

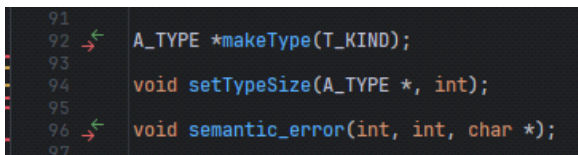


setFunctionDeclaratorSpecifier()의 해당 부분도 주석처리 하였다.

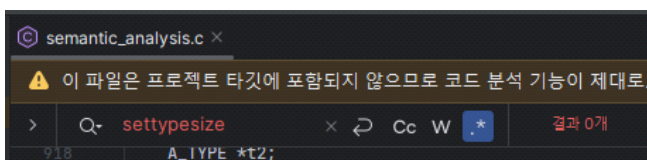
2-2. semantic_analysis.c(h)

부록 6을 이전 과제처럼 휴대폰의 ocr 기능으로 스캔하되 다음과 같은 수정을 하였다.

- setTypeSize()



semantic_analysis.h(부록 6의 초반)에는 함수가 선언되어 있었으나, 프로젝트 그 어디에도, 부록에도, 이의 정의는 존재하지 않았다(사진처럼 ide로 헤더파일을 보았을 때, 왼쪽에 화살표가 없으면 정의되지 않았다는 뜻).



semantic_analysis.c를 보니 setTypeSize()가 사용되는 곳이 단 하나도 없었기에, 단순히 헤더파일에서 이를 주석처리 하기로 한다.

- atof()

```
kh@ThinkPad-T16g2: ~  
ATOF(3) Linux Programmer's Manual  
NAME  
    atof - convert a string to a double  
SYNOPSIS  
    #include <stdlib.h>  
  
    double atof(const char *nptr);
```

atof는 c언어의 stdlib의 함수이다. 현재 개발 환경은 64비트이기에 atof는 double형을 반환한다.

```
20  
21 float atof();  
22  
23 void sema Conflicting types for 'atof'  
24 previous declaration is here  
25 void set_  
26 함수 'atof' 안전한 삭제 Alt+Sh  
27 int put_l
```

반환형이 충돌이 난다는 경고가 발생한다.

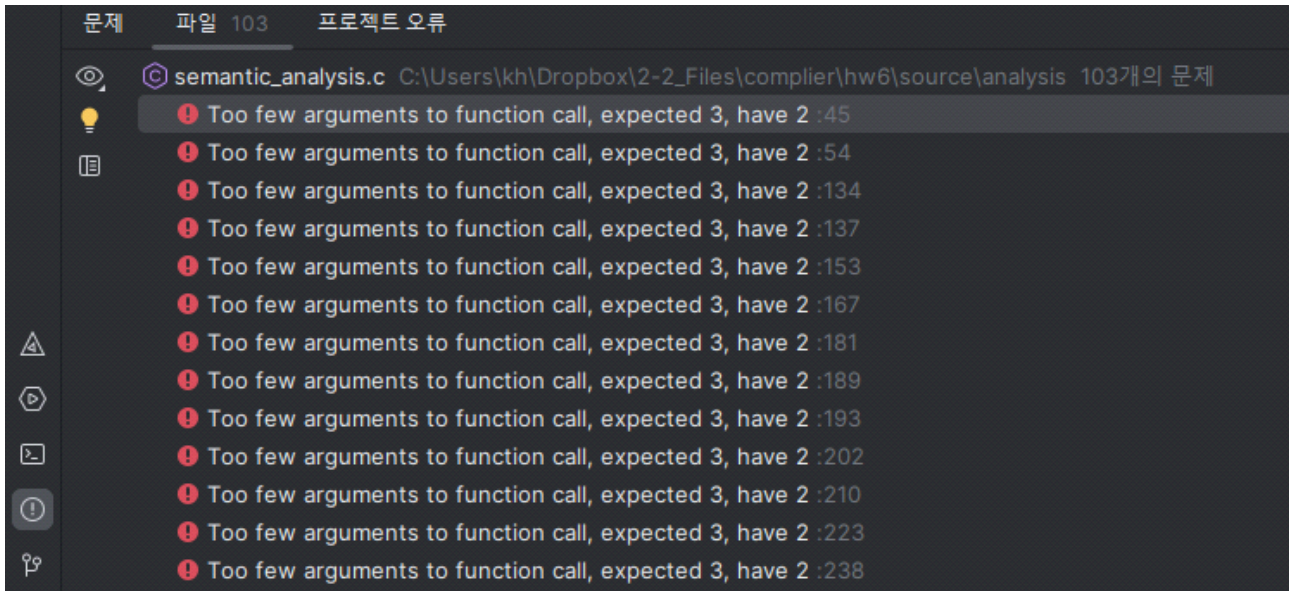
```
13  
14 float myAtof(char *s){  
15     double f = atof(s);  
16  
17     return (float) f;  
18 }  
19
```

```
break;  
case N_EXP_FLOAT_CONST:  
    lit.type = float_type;  
    lit.value.f = myAtof(node->clink);  
    node->clink = put_literal(lit, node->  
    result = float_type;
```

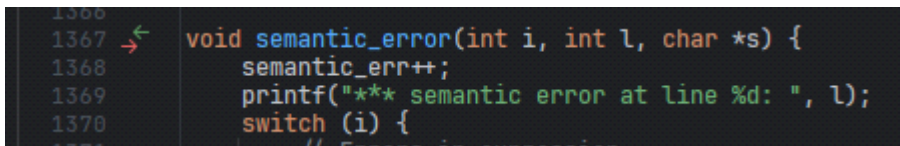
```
break;  
case N_EXP_FLOAT_CONST:  
    result.type = float_type;  
    result.value.f = myAtof(node->clink);  
    break;  
case N_EXP_STRING_LITERAL:
```

atof는 sem_expression(), getTypeAndValueOfExpression()의 두 곳에서 사용된다. double을 float으로 캐스팅해주는 간단한 함수인 myAtof()로 대체해주었다.

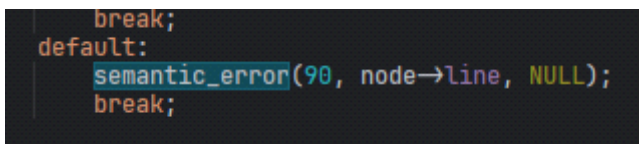
- semantic_error()



이전 과제의 syntax_error()처럼 매 semantic_error() 호출마다 에러가 났다.

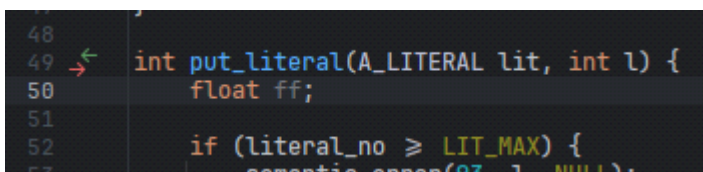


마지막 파라미터인 string이 전달되지 않는 case가 많기 때문이다.

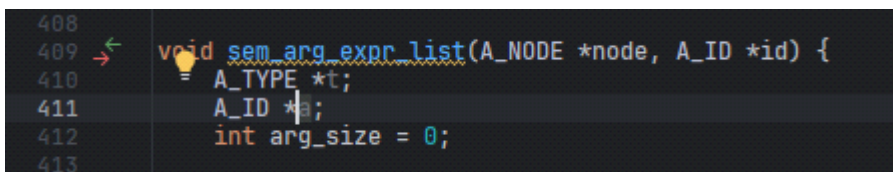


해당하는 경우 전부 마지막 파라미터로 NULL을 넣게끔 수정했다.

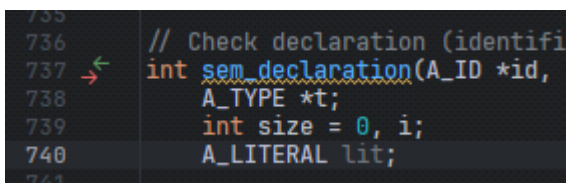
- 사용되지 않는 지역변수 삭제



put_literal()의 ff



sem_arg_expr_list()의 a

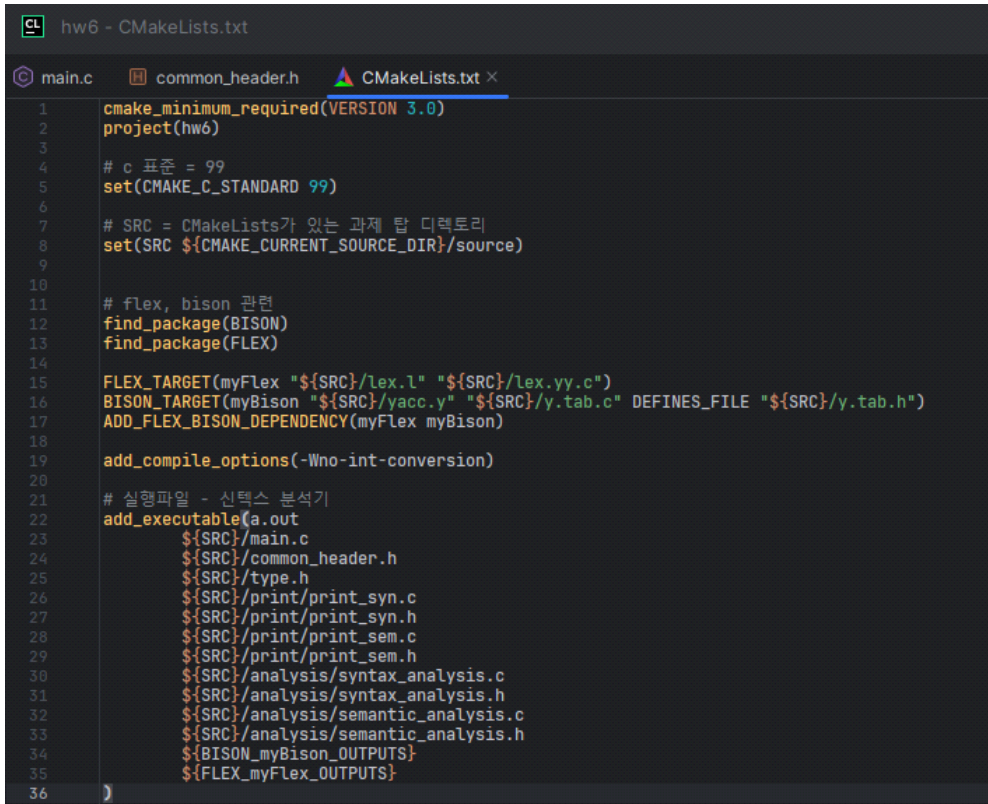


sem_declaration()의 lit가 해당 함수 내에서 전혀 사용되지 않아서 삭제했다.

- semantic_error()

이는 수정사항이 아니긴 하나 일단 적는다. 이전 과제의 syntax analysis시 오류가 발생하면 올바른 syntax tree를 구성하지 못하기에, syntax_error()가 호출되면 그 즉시 exit(1)을 실행했다. 그러나 syntax analysis를 통과하고 발생한 semantic error는 문법적으로는 맞는 것이기에, 추후에 프로그래머에게 어떤 위치에 어떤 수정을 하라 알려줘야 한다. 따라서 semantic_error는 exit()을 하지 않을 것이다.

2-3. CMakeLists.txt



```
1 cmake_minimum_required(VERSION 3.0)
2 project(hw6)
3
4 # c 표준 = 99
5 set(CMAKE_C_STANDARD 99)
6
7 # SRC = CMakeLists가 있는 과제 탭 디렉토리
8 set(SRC ${CMAKE_CURRENT_SOURCE_DIR}/source)
9
10
11 # flex, bison 관련
12 find_package(BISON)
13 find_package(FLEX)
14
15 FLEX_TARGET(myFlex "${SRC}/lex.l" "${SRC}/lex.yy.c")
16 BISON_TARGET(myBison "${SRC}/yacc.y" "${SRC}/y.tab.c" DEFINES_FILE "${SRC}/y.tab.h")
17 ADD_FLEX_BISON_DEPENDENCY(myFlex myBison)
18
19 add_compile_options(-Wno-int-conversion)
20
21 # 실행파일 - 신택스 분석기
22 add_executable(a.out
23     ${SRC}/main.c
24     ${SRC}/common_header.h
25     ${SRC}/type.h
26     ${SRC}/print/print_syn.c
27     ${SRC}/print/print_syn.h
28     ${SRC}/print/print_sem.c
29     ${SRC}/print/print_sem.h
30     ${SRC}/analysis/syntax_analysis.c
31     ${SRC}/analysis/syntax_analysis.h
32     ${SRC}/analysis/semantic_analysis.c
33     ${SRC}/analysis/semantic_analysis.h
34     ${BISON_myBison_OUTPUTS}
35     ${FLEX_myFlex_OUTPUTS}
36 )
```

새롭게 추가된 파일과 디렉터리 구조 변경에 따라서 적절히 구성하였다.

3. 빌드

이전 과제처럼 ide에서 CMake를 사용하여 빌드하였다.

오류 없이 빌드가 되었다.

4. 테스트

양이 많아서 편의를 위해 다른 파일에 적겠습니다.

5. 소스코드

부록 없이 따로 파일을 첨부하겠습니다.